7

Bitcoin in Practice

In this chapter, we will examine how the Bitcoin protocol works, as well as some advanced and modern Bitcoin protocols that have been developed to address limitations in the original Bitcoin protocol.

In this chapter, we will cover:

- Bitcoin in the real world
- Bitcoin payments
- Innovation in Bitcoin
- Bitcoin client installation
- Experimenting further with `bitcoin-cli`
- Bitcoin programming

Before we talk about the specifics of Bitcoin, let's briefly discuss the philosophy behind it, go over the official definition of Bitcoin, and consider Bitcoin from a user's perspective.

Bitcoin in the real world

For people with a libertarian ideology, Bitcoin is a platform that can be used instead of banks. However, some think that due to regulations, Bitcoin may become another institution that cannot be trusted. The original idea behind Bitcoin was to develop an e-cash system that requires no trusted third party and where users can be anonymous. If regulations require checks like **Know Your Customer** (**KYC**) and detailed information about business transactions to facilitate the regulatory process, then it might be too much information to share. As a result, Bitcoin may not remain attractive to some entities.

The regulation of Bitcoin is a controversial subject. As much as it is a libertarian's dream, law enforcement agencies, governments, and banks are proposing various regulations to control it.

Interested readers can read more about the regulation of Bitcoin and other relevant news and activities at `https://cointelegraph.com/tags/Bitcoin-regulation`.

At this point, the question arises that if Bitcoin is under so much pressure from regulatory bodies, then how has it managed to grow so significantly? The simple answer is due to its decentralized and trustless nature. In this context, the term *trustless* refers to the distribution of trust between users, rather than a central entity. No single entity can control this network, and even if some entities try to enforce some regulations, they can only go so far because the network is owned collectively by its users instead of a single entity. It is also protected by its PoW mechanism, which thwarts any adversarial attacks on the network. Moreover, the anonymity of the founder of Bitcoin has also played a role in its success.

The growth of Bitcoin is also due to the so-called **network effect**. Also called **demand-side economies of scale**, this is a concept that means that the more users use the network, the more valuable it becomes. Over time, an exponential increase has been seen in Bitcoin network growth. This increase in the number of users is mostly driven by financial incentives. Also, the scarcity of Bitcoin and its built-in inflation control mechanism gives it value, as there are only 21 million Bitcoins that can ever be mined. Also, the miner reward halves every four years, which increases scarcity, and consequently, the demand increases even more.

Despite its overall success, there are some concerns regarding Bitcoin as well. Bitcoin's **ESG** (**Environmental, Social, and Governance**) impact is also a concern. The main arguments include mining centralization where some powerful miners have most of the network (hash power), high energy consumption, and centralized development and governance. Moreover, Bitcoin volatility is another concern that is seen as a barrier to its day-to-day adoption.

However, despite the regulatory pressure, some limitations, and it even being made illegal in some countries, Bitcoin has gained significant acceptance in some parts of the world. For example, Bitcoin is legal tender in El Salvador and the Central African Republic.

In the next section, we will see how the Bitcoin network looks from a user's point of view—how a transaction is made, how it propagates from the user to the network, and how transactions are verified and finally accumulated in blocks.

Bitcoin payments

Having explored the various components of Bitcoin's architecture and design in the previous chapter, the following example will help you to understand how a payment transaction via the Bitcoin network looks from the end user's perspective. There are several steps involved in this process. In this example, we are using the **Blockchain wallet** for mobile devices.

The steps are described as follows:

1. First, either the payment is requested from a user, or the sender initiates a transfer to send money to another user. In both cases, the Bitcoin address of the beneficiary is required to be sent via an appropriate communication mechanism.
2. The sender either enters the receiver's address or scans the generated QR code that has the Bitcoin address, amount, and an optional description encoded in it. The wallet application recognizes this QR code and decodes it into something like:

```
"Please send <amount> BTC to address <receiver's Bitcoin address>".
```

With actual values, this will look like the following:

```
"Please send 0.00033324 BTC to address
1JzouJCVmMQBmTcd8K4Y5BP36gEFNn1ZJ3".
```

3. In the wallet application of the sender, this transaction is constructed, digitally signed using the private key of the sender, then broadcast to the Bitcoin network.
4. Bitcoin transactions are serialized for transmission over the network and encoded in hex format. As an example, this transaction is also shown in raw serialized hex format as follows:

```
01000000017d3876b14a7ac16d8d550abc78345b6571134ff173918a096ef
90ff0430e12408b0000006b483045022100de6fd8120d9f142a82d5da9389
e271caa3a757b01757c8e4fa7afbf92e74257c02202a78d4fbd52ae9f3a00
83760d76f84643cf8ab80f5ef971e3f98ccba2c71758d012102c16942555f
5e633645895c9affcb994ea7910097b7734a6c2d25468622f25e12ffffff
ff022c820000000000001976a914c568ffeb46c6a9362e44a5a49deaa6ea
b05a619a88acc06c010000000000001976a9149386c8c880488e80a6ce8f18
6f788f3585f74aee88ac00000000
```

5. Once the QR code is decoded, the transaction will appear in the wallet. There are a number of parameters required for a transaction to work, such as `From`, `To`, `BTC`, and `Fee`. Bitcoin network fees ensure that your transaction will be included by miners in the block.
6. This transaction will be picked up by miners to be verified for legitimacy and included in the block. A confirmation will appear as soon as the transaction is verified, included in the candidate or proposed block, and mined.
7. Usually, at this point, users wait for up to six confirmations to be received before a transaction is considered final; however, a transaction can be considered final at the previous step. Confirmations serve as an additional mechanism to ensure that there is probabilistically a very low chance of a transaction being reverted, but otherwise, once a mined block is finalized and announced, the transactions within that block are final at that point.
8. The appropriate fee will be deducted from the original value to be transferred and will be paid to the miner who has included it in the block for mining.

In the transaction flow just described, a total payment of 0.00033324 BTC left the sender's address, 0.001267 BTC of which was paid to the receiver's address. A fee of 0.00010622 was deducted from the transaction as a mining fee.

Transactions are not encrypted and are publicly visible on the blockchain. We can see these details using a blockchain explorer, such as blockchain.info: `https://www.blockchain.com/btc/address/1JzouJCVmMQBmTcd8K4Y5BP36gEFNn1ZJ3`

Moreover, a view of various attributes of the transaction is available here: `https://www.blockchain.com/btc/tx/d28ca5a59b2239864eac1c96d3fd1c23b747f0ded8f5af0161bae8a616b56a1d`

Bitcoin can be accepted as payment using various techniques. It is increasingly being accepted as a payment method by many online merchants and e-commerce websites. There are a number of ways in which buyers can pay a business that accepts Bitcoin. For example, in an online shop, Bitcoin merchant solutions can be used, whereas in traditional, physical shops, **Point of Sale** (**POS**) terminals and other specialized hardware can be used. Customers can simply scan the QR barcode with the seller's payment URI in it and pay using their mobile devices. Bitcoin URIs allow users to make payments by simply clicking on links or scanning QR codes. A **Uniform Resource Identifier** (**URI**) is a string that represents the transaction information. The QR code can be displayed near the point of sale terminal, which can be decoded by wallets.

Various payment solutions, such as the 34 Bytes Bitcoin POS terminal, are available commercially. Generally, these solutions work by following these steps:

1. The salesperson enters the amount of money to be charged in fiat currency, for example, US dollars.
2. Once the value is entered into the system, the terminal prints a receipt with a QR code on it and other relevant information, such as the amount to be paid.
3. The customer can then scan this QR code using their mobile Bitcoin wallet to send the payment to the Bitcoin address of the seller embedded within the QR code.
4. Once the payment is received at the designated Bitcoin address, a receipt is printed out as physical evidence of the sale.

Bitcoin payment processors are offered by many online service providers. This allows integration with e-commerce websites to facilitate Bitcoin payments. These payment processors can be used to accept Bitcoin as payment. Some service providers also allow the secure storage of bitcoins, for example, **BitPay** (`https://bitpay.com`). Another example is the Bitcoin merchant solutions available at `https://www.bitcoin.com/merchant-solutions`.

Bitcoin, and blockchain technology in general, is continuously evolving, and we'll discuss some of the most relevant ideas next.

Innovation in Bitcoin

Bitcoin has undergone many changes and is still evolving into a more and more robust and better system by addressing various weaknesses in the system. Performance has been a topic of hot debate among Bitcoin experts and enthusiasts for many years. As such, various proposals have been made in the last few years to improve Bitcoin performance, resulting in greater transaction speed, increased security, payment standardization, and overall performance improvement at the protocol level.

These improvement proposals are usually made in the form of **Bitcoin Improvement Proposals** (**BIPs**) or fundamentally new versions of Bitcoin protocols, resulting in new networks altogether. Some of the changes proposed can be implemented via a soft fork, but a few need a hard fork and, as a result, give birth to a new currency.

Bitcoin improvement proposals

These documents, also referred to as **BIPs**, are used to propose improvements or inform the Bitcoin community about the improvements suggested, the design issues, or some aspects of the Bitcoin ecosystem. There are three types of BIPs:

- **Standard BIP**: Used to describe the major changes that have a major impact on the Bitcoin system; for example, block size changes, network protocol changes, or transaction verification changes.
- **Process BIP**: A major difference between standard and process BIPs is that standard BIPs cover protocol changes, whereas process BIPs usually deal with proposing a change in a process that is outside the core Bitcoin protocol. These are implemented only after a consensus among Bitcoin users.
- **Informational BIP**: These are usually used to just advise or record some information about the Bitcoin ecosystem, such as design issues.

Various BIPs have been proposed and finalized to introduce and standardize Bitcoin payments. Most notably, BIP70 (secure payment protocol) describes the protocol for secure communication between a merchant and customers. This protocol uses X.509 certificates for authentication and runs over HTTP and HTTPS. There are three messages in this protocol: **PaymentRequest**, **Payment**, and **PaymentACK**. The key features of this proposal are defense against man-in-the-middle attacks and secure proof of payment.

Man-in-the-middle attacks can result in a scenario where the attacker is sitting between the merchant and the buyer, and it would seem to the buyer that they are talking to the merchant, but in fact, the man in the middle is interacting with the buyer instead of the merchant. This can result in the manipulation of the merchant's Bitcoin address to defraud the buyer.

Several other BIPs, such as BIP71 (Payment Protocol MIME types) and BIP72 (URI extensions for Payment Protocol), have also been implemented to standardize payment schemes to support BIP70 (Payment Protocol).

Another innovative development is the Lightning Network. It is a solution for scalable off-chain instant payments. It was introduced in early 2016 and allows off-blockchain payments. This network makes use of payment channels that run off the blockchain, which allows for the greater speed and scalability of Bitcoin.

A paper is available at `https://lightning.network/` and those of you who are interested are encouraged to read the paper to understand the theory and rationale behind this invention.

Now that we've provided this brief discussion on Bitcoin's improvement and evolution, let's look at some of the excellent ideas that have emerged from research and innovation efforts related to Bitcoin.

Advanced protocols

In this section, we'll introduce various advanced protocols that have been suggested or implemented for improving the Bitcoin protocol. For example, transaction throughput is one of the critical issues that need a solution. The Bitcoin network can only process approximately three to seven transactions per second, which is a tiny number compared to other financial networks. For example, the Visa network can process approximately, on average, 24,000 transactions per second. PayPal can process approximately 200 transactions per second, whereas Ethereum can process up to, on average, 20. As the Bitcoin network has grown exponentially over the last few years, these issues have started to grow even further. The difference in processing speed is also shown in the following graph, which shows the scale of difference between Bitcoin and other networks' transaction speeds. The graph uses a logarithmic scale, which demonstrates the vast difference between the networks' transaction speeds:

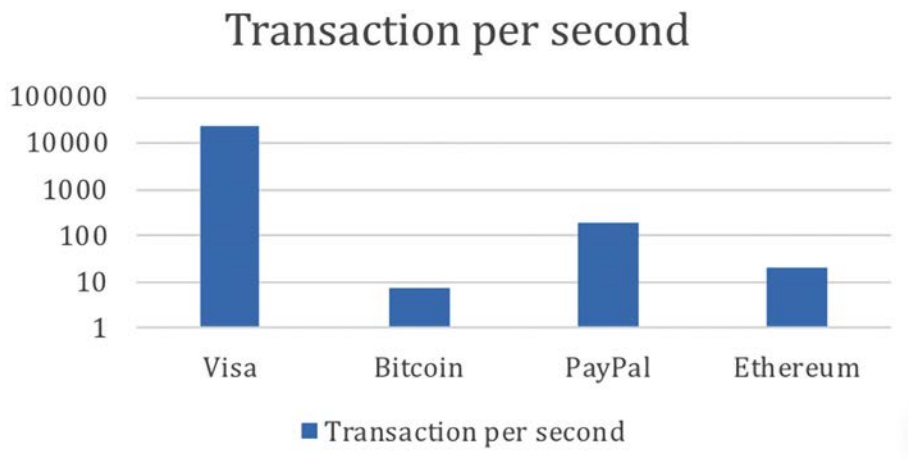


Figure 7.1: Bitcoin transaction speed compared to other networks (on a logarithmic scale)

Also, security issues such as transaction malleability are of real concern and can result in denial of service. Various proposals have been made to improve the Bitcoin proposal to address various weaknesses. A selection of these proposals will be presented here.

Segregated Witness

The **SegWit** or **Segregated Witness** is a soft fork-based upgrade of the Bitcoin protocol that addresses weaknesses such as throughput and security in the Bitcoin protocol. SegWit offers several improvements, as listed here:

- A fix for transaction malleability due to the separation of signature data from transactional data. In this case, it is no longer possible to modify the transaction ID because it is no longer calculated based on the signature data present within the transaction.
- By segregating the signature data and transaction data, lightweight clients do not need to download the transactions with all signatures unnecessarily. The transactions can be verified without useless signatures, which allows for increased bandwidth efficiency.

- Reduction in transaction signing and verification times, which results in faster transactions. A new transaction hashing algorithm for signature verification has been introduced and is detailed in BIP0143 (`https://en.bitcoin.it/wiki/BIP_0143`). Due to this change, the verification time grows linearly with the number of inputs instead of in a quadratic manner, resulting in a quicker verification time.
- A script versioning capability, which allows for easier script language upgrades. The version number is prefixed to the locking scripts to depict the version. This change allows upgrades and improvements to be made to the scripting language, without requiring a hard fork, by just increasing the version number of the script.
- Increased block size by introducing a weight limit instead of a size limit on the block and the removal of signature data. This concept will be explained in more detail shortly.
- An improved address format, also called a "bc1 address," which is encoded using the **Bech32** mechanism instead of base58. This improvement allows for better error detection and correction. Also, all characters are lowercase, which helps with readability. Moreover, this helps with distinguishing between legacy transactions and SegWit transactions. More information is available at this link: `https://en.bitcoin.it/wiki/Bech32`.

SegWit was proposed in BIP141, BIP143, BIP144, and BIP145. It was activated on Bitcoin's main network on August 24 2017 at block number 481824. The key idea behind SegWit is the separation of signature data from transaction data (that is, a transaction Merkle tree), which results in the size of the transaction being reduced. This change allows the block size to increase up to 4 MB in size. However, the practical limit is between 1.6 MB and 2 MB. Instead of a hard size limit of 1 MB blocks, SegWit introduced a new concept of a block weight limit.

The block weight is a new restriction mechanism where each transaction has a weight associated with it. This **weight** is calculated on a per-transaction basis. The formula used to calculate it is:

$$\textit{Weight} = (\textit{Transaction size without witness data}) \text{ x } 3 + (\textit{Transaction size})$$

Blocks can have a maximum of four million weight units. As a comparison, a byte in a legacy 1 MB block is equivalent to 4 weight units, but a byte in a SegWit block weighs only 1 weight unit. This modification immediately results in increased block capacity.

To spend an **Unspent Transaction Output** (**UTXO**) in Bitcoin, a valid signature needs to be provided. In the pre-SegWit scenario, this signature is provided within the locking script, whereas in SegWit this signature is not part of the transaction and is provided separately.

There are four types of transactions introduced by SegWit. These types are:

1. **Pay to Witness Public Key Hash (P2WPKH)**: This type of script is similar to the usual P2PKH, but the crucial difference is that the transaction signature used as a proof of ownership in ScriptSig is moved to a separate structure known as the "witness" of the input. The signature is the same as P2PKH but is no longer part of ScriptSig; it is simply empty. The PubKey is also moved to the witness field.

This script is identified by a 20-byte hash. The ScriptPubKey is modified to a simpler format, as shown here:

- P2PKH ScriptPubKey:

```
OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

- P2WPKH ScriptPubKey:

```
OP_0 <pubKeyHash>
```

2. **Pay to Script Hash – Pay to Witness PubKey Hash** (**P2SH-P2WPKH**): This is a mechanism introduced to make SegWit transactions backward-compatible. This is made possible by nesting the P2WPKH inside the usual P2SH.
3. **Pay to Witness Script Hash** (**P2WSH**): This script is similar to legacy P2SH but the signature and redeem script are moved to the separate witness field. This means that ScriptSig is simply empty. This script is identified by a 32-byte SHA-256 hash. P2WSH is a simpler script compared to P2SH and has just two fields. The ScriptPubKey is modified as follows:

   P2SH ScriptPubKey:

```
OP_HASH160 <pubKeyHash> OP_EQUAL
```

   P2WSH ScriptPubKey:

```
OP_0 <pubKeyHash>
```

4. **Pay to Script Hash – Pay to Witness Script Hash** (**P2SH-P2WSH**): Similar to P2SH-P2WPKH, this is a mechanism that allows backward-compatibility with legacy Bitcoin nodes.

SegWit adoption is still in progress as not all users of the network agree or have started to use SegWit.

Next, we'll introduce some other innovative ideas in the Bitcoin space. Not only has the original Bitcoin evolved quite significantly since its introduction, but there are also new blockchains that are either forks of Bitcoin or novel implementations of the Bitcoin protocol with advanced features.

Bitcoin Cash

Bitcoin Cash (**BCH**) increases the block limit to 8 MB. This change immediately increases the number of transactions that can be processed in one block to a much larger number compared to the 1 MB limit in the original Bitcoin protocol. It uses **Proof of Work** (**PoW**) as a consensus algorithm, and mining hardware is still ASIC-based. The block interval is changed from 10 minutes to 10 seconds and up to 2 hours. It also provides replay protection and wipe-out protection, which means that because BCH uses a different hashing algorithm, it prevents it from being replayed on the Bitcoin blockchain. It also has a different type of signature compared to Bitcoin to differentiate between two blockchains.

The BCH wallet and relevant information are available on their website: `https://www.bitcoincash.org`.

Bitcoin Unlimited

Bitcoin Unlimited increases the size of the block without setting a hard limit. Instead, miners come to a consensus on the block size cap over a period of time. Other concepts such as extremely thin blocks and parallel validation have also been proposed in Bitcoin Unlimited.

Its client is available for download at `https://www.bitcoinunlimited.info`.

Extremely thin blocks allow for faster block propagation between Bitcoin nodes. In this scheme, the node requesting blocks sends a `getdata` request, along with a bloom filter, to another node. The purpose of this bloom filter is to filter out the transactions that already exist in the **memory pool** (**mempool**) of the requesting node. The node then sends back a **thin block** only containing the missing transactions. This fixes an inefficiency in Bitcoin whereby transactions are regularly received twice—once at the time of broadcast by the sender and then again when a mined block is broadcast with the confirmed transaction.

Parallel validation allows nodes to validate more than one block, along with new incoming transactions, in parallel. This mechanism contrasts with Bitcoin, where a node, during its validation period after receiving a new block, cannot relay new transactions or validate any blocks until it has accepted or rejected the block.

Bitcoin Gold

This proposal has been implemented as a hard fork since block 491407 of the original Bitcoin blockchain. Being a hard fork, it resulted in a new blockchain, named Bitcoin Gold. The core idea behind this concept is to address the issue of mining centralization, which has hurt the original Bitcoin idea of decentralized digital cash, whereby more hash power has resulted in a power shift toward miners with more hashing power. Bitcoin Gold uses the Equihash algorithm as its mining algorithm instead of PoW; hence, it is inherently ASIC resistant and uses GPUs for mining.

Bitcoin Gold and relevant information are available at `https://bitcoingold.org`.

Taproot

Taproot is a significant upgrade to the Bitcoin protocol. It was activated at block 709,632. It improves transaction speed, privacy, and scalability. It also allows smart contracts on the Bitcoin network. Taproot includes several components, which we explain next:

- **Schnorr signatures**, which provide signature aggregation resulting in privacy and efficiency. Schnorr signatures are more secure than ECDSA.

- **Merkelized Alternative Script Tree** (**MAST**), which compresses complex bitcoin transactions into a single hash that reduces the transaction fee and memory usage. MAST allows enumerating distinct spending conditions separately. This allows funds to be spent by satisfying any one of the scripts.

  In MAST each script lives in a leaf on the Merkle tree. When funds are received, they are locked to the Merkle tree root. To spend the funds, a single leaf's script is revealed, satisfying the spending conditions mandated in the leaf. Here, a Merkle proof proves its inclusion in the tree. This way, other spending conditions that are not relevant are kept private. It also means that multiple different spending conditions can be encoded in many leaves.

- **Pay2Taproot** (**P2TR**) is a new type of script. This script combines the Schnorr signature and MAST in a single transaction. The Tapscript scripting language is used to enable various types of new transactions. It is like the Script language (the original Bitcoin script), but with some changes. The key change is the introduction of `OP_CHECKSIGADD` opcode, which enables the aggregation of signatures by utilizing Schnorr signatures.

  Tapscript also enables easier future soft fork upgrades by using the new `OP_SUCCESS` opcode. In practice, P2TR addresses combine signatures and scripts by placing the MAST of scripts in a public key. This means that the same funds can be spent either with a usual plain signature that corresponds to that public key (the old way) or through one of the scripts encoded in the MAST.

The Taproot upgrade is composed of three BIPs: BIP340 (BIP – Schnorr), BIP341 (BIP – Taproot), and BIP342 (BIP – Tapscript).

Extended protocols on top of Bitcoin

Several protocols, as discussed in the following sections, have been proposed and implemented on top of Bitcoin to enhance and extend the Bitcoin protocol, as well as to be used for various other purposes instead of just as a virtual currency.

Colored coins

Colored coins are a set of methods that have been developed to represent digital assets on the Bitcoin blockchain. Coloring a bitcoin refers colloquially to updating it with some metadata representing a digital asset (smart property). The coin still works and operates as a Bitcoin, but additionally carries some metadata that represents some assets. This can be some information related to the asset, some calculations related to transactions, or any arbitrary data. This mechanism allows issuing and tracking specific bitcoins. Metadata can be recorded using Bitcoin's `OP_RETURN` opcode or optionally in multi-signature addresses. The metadata can also be encrypted if required to address any privacy concerns. Some implementations also support the storage of metadata on publicly available torrent networks, which means that virtually unlimited amounts of metadata can be stored. Usually, these are JSON objects representing various attributes of the colored coin. Moreover, smart contracts are also supported.

Colored coins can be used to represent a multitude of assets, including, but not limited to, commodities, certificates, shares, bonds, and voting. It should also be noted that to work with colored coins, a wallet that interprets colored coins is necessary, and normal Bitcoin wallets will not work. Normal Bitcoin wallets will not work because they cannot differentiate between **colored coins** and **not colored coins**.

The idea of colored coins is very appealing as it does not require any modifications to be made to the existing Bitcoin protocol, and they can also make use of the already existing secure Bitcoin network. In addition to the traditional representation of digital assets, there is also the possibility of creating smart assets that behave according to the parameters and conditions defined for them. These parameters include time validation, restrictions on transferability, and fees. This opens the possibility of creating smart contracts, which we will discuss in *Chapter 8*, *Smart Contracts*.

A significant use case is the issuance of financial instruments on the blockchain. This will ensure low transaction fees, valid and mathematically secure proof of ownership, fast transferability without requiring some intermediary, and instant dividend payouts to investors.

There were a few services available online that used to provide colored coins services, but they are no longer active. However, check this link for some more information on this: *Colu* by *Coinprism* (`https://en.bitcoin.it/wiki/Coinprism`).

Counterparty

This is another service that can be used to create custom tokens that act as a cryptocurrency and can be used for various purposes, such as issuing digital assets on top of the Bitcoin blockchain. This is quite a robust platform and runs on Bitcoin blockchains at its core, but has developed its client and other components so that they support issuing digital assets. The architecture consists of the following components:

1. **Counterparty server**: This is the reference client and implements the core counterparty protocol. It is a combination of `counterparty-lib` and `counterparty-cli`.
2. **Counter block**: This component provides services in addition to the Counterparty server.
3. **Counter wallet**: This is a web wallet for Bitcoin and **Counterparty Coins** (**XCPs**).
4. **armory_utxsvr**: This is a service used for offline armory transactions.

Counterparty works based on the same idea as colored coins by embedding data into regular Bitcoin transactions but provides a much more productive library and a set of powerful tools to support the handling of digital assets. This embedding is also called **embedded consensus** because the counterparty transactions are embedded within Bitcoin transactions. The method of embedding the data is by using the `OP_RETURN` opcode in Bitcoin.

The currency produced and used by Counterparty is known as XCP and is used by smart contracts as the fee for running the contract. At the time of writing, its price is 1.03 USD. XCPs were created by using the PoB method discussed previously.

Counterparty allows the development of smart contracts on Ethereum using the Solidity language and allows interaction with the Bitcoin blockchain. To achieve this, BTC Relay is used to provide interoperability between Ethereum and Bitcoin. This is a clever concept where Ethereum contracts can talk to the Bitcoin blockchain and transactions through BTC Relay. The relayers (the nodes that are running BTC Relay) fetch the Bitcoin block headers and relay them to a smart contract on the Ethereum network that verifies the PoW. This process verifies that a transaction has occurred on the Bitcoin network.

BTC Relay is available at `http://btcrelay.org/`.

Technically, this is an Ethereum contract that can store and verify Bitcoin block headers, just like Bitcoin simple payment verification lightweight clients do, by using bloom filters. SPV clients were discussed in detail in the previous chapter. This idea can be visualized with the following diagram:

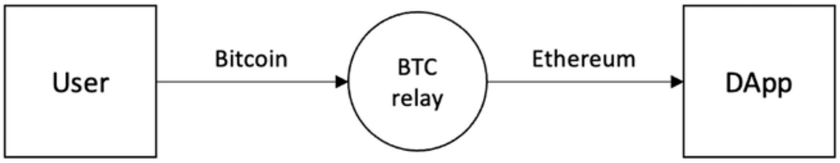


Figure 7.2: BTC relay concept

Counterparty is available at `http://counterparty.io/`.

Now, we will move on to a different topic, which explains how altcoins are developed, how they work, and how difficult it is to create a new coin.

Altcoins from Bitcoin

By definition, an altcoin is generated in the case of a hard fork. Altcoins must be able to attract new users, trades, and miners; otherwise, the currency will have no value. Currency gains its value, especially in the virtual currency space, due to the network effect and its acceptability by the community. If a coin fails to attract enough users, then soon, it will be forgotten. Users can be attracted by providing an initial amount of coins, which can be achieved by using various methods. There is, however, a risk that if the new coin does not perform well, then its initial investment may be lost.

Methods of providing an initial number of altcoins are as follows:

1. **Create a new blockchain**: Altcoins can create a new blockchain and allocate coins to initial miners, but this approach is now unpopular due to many scam schemes, or *pump-and-dump* schemes, where initial miners made a profit with the launch of a new currency and then disappeared.
2. **Proof of Burn** (**PoB**): Another approach to allocating initial funds to a new altcoin is PoB, also called a one-way peg or price ceiling. In this method, users permanently destroy a certain quantity of bitcoins in proportion to the number of altcoins to be claimed. For example, if 10 bitcoin are destroyed, then altcoins can have a value no greater than the bitcoins that were destroyed. This means that bitcoins are converted into altcoin by burning them.
3. **Proof of ownership**: Instead of permanently destroying bitcoins, an alternative method is to prove that users own a certain number of bitcoins. This proof of ownership can be used to claim altcoins by tethering altcoin blocks to Bitcoin blocks. For example, this can be achieved through merged mining in which, effectively, Bitcoin miners can mine altcoin blocks while mining for bitcoins without any extra work. Merged mining will be explained later in this chapter.
4. **Pegged sidechains**: Sidechains, as the name suggests, are blockchains separate from the Bitcoin network, but Bitcoin can be transferred to them. Altcoins can also be transferred back to the Bitcoin network. This concept is called a **two-way peg**.

We have now covered some new blockchains and implementations of the Bitcoin protocol. The next section introduces Bitcoin client installation and a basic overview of various APIs and tools that are available for developing Bitcoin applications and interacting with the Bitcoin blockchain.

Bitcoin client installation

The Bitcoin Core client can be installed from `https://bitcoin.org/en/download`. This is available for different architectures and platforms, ranging from x86 Windows to ARM Linux.

We will discuss a few topics relating to Bitcoin's installation and setup. We'll begin by discussing the different Bitcoin clients available and their associated tools, which enable you to run and manage the Bitcoin client and interact with the Bitcoin blockchain.

Types of clients and tools

There are different types of Bitcoin Core clients and relevant tools. A Bitcoin client is a piece of software that is responsible for generating private/public key pairs and facilitates Bitcoin payments using the Bitcoin blockchain. In addition, a client can implement a full synchronization function with a blockchain or choose to only implement basic wallet functionality or simple payment verification. A client can also provide other useful functions such as network monitoring, secure storage of keys, and user-friendly interfaces for interaction with the Bitcoin blockchain. Some of the core elements of the Bitcoin Core client and associated tools are as follows:

- `bitcoind`: This is the core client software that runs as a daemon (as a service), and it provides the JSON-RPC interface.

- `bitcoin-cli`: This is the command-line feature-rich tool for interacting with the Bitcoin daemon; the daemon then interacts with the blockchain and performs various functions. `bitcoin-cli` only calls JSON-RPC functions and does not perform any actions on its own on the blockchain.
- `bitcoin-qt`: This is the Bitcoin Core client GUI. When the wallet software starts up, first, it verifies the blocks on the disk and then starts the block synchronization process. The verification process is not specific to the `bitcoin-qt` client; it is performed by the `bitcoind` client as well.

There are also other clients available, such as **btcd**, which is a full-node Bitcoin client written in Golang. It is available at `https://github.com/btcsuite/btcd`.

Setting up a Bitcoin node

In this section, we will explore how we can set up a Bitcoin node in order to interact with the Bitcoin network and interact with a Bitcoin node using the command-line interface.

The Bitcoin Core software is available at `https://bitcoin.org/en/download`. You can download the software and install it according to the instructions provided, which are quite straightforward.

Alternatively, you can download the Bitcoin source code and compile it manually to produce binaries, which we'll cover next.

Setting up the source code

The Bitcoin source code can be downloaded and compiled if users wish to use the Bitcoin code, for learning purposes, or just want to produce binaries manually. The `git` command can be used to download the Bitcoin source code:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
```

Change the directory to `bitcoin`:

```
$ cd bitcoin
```

After the preceding steps are completed, the code can be compiled:

```
$ ./autogen.sh
$ ./configure.sh
$ make
$ sudo make install
```

Note that the `make` command shown here may take around 30 minutes to complete, depending on the speed of your computer.

Setting up bitcoin.conf

The `bitcoin.conf` file is a configuration file that is used by the Bitcoin Core client to save configuration settings. All command-line options for the `bitcoind` client, except for the `-conf` switch, can be set up in the configuration file, and when `bitcoin-qt` or `bitcoind` starts up, it will take the configuration information from that file.

In Linux systems, this is usually found in `$HOME/.bitcoin/`, but it can also be specified in the command line using the `-conf=<file>` switch in the `bitcoind` core client software. The configuration file can be generated using the tool available here: `https://github.com/bitcoin/bitcoin/tree/master/contrib/devtools#gen-bitcoin-confsh`.

Now that we have set up the Bitcoin client, let's see how to start up the Bitcoin client for use.

Starting up a node in the testnet

The Bitcoin node can be started in a test blockchain network (testnet) if you want to test the Bitcoin network and run some experiments. This is a completely alternative test network used for experimentation. It is a faster network compared to the main network and has relaxed rules for mining and transactions.

The key differences between the mainnet and the testnet are shown here:

Component	**Mainnet**	**Testnet**
Listening port	TCP 8333	TCP 18333
RPC connection port	TCP 8332	TCP 18332
DNS seeds are different for bootstrapping	Mainnet-specific	Testnet-specific
A different `ADDRESSVERSION` field in addresses to ensure testnet addresses do not work on Bitcoin's mainnet	0x00	0x6F
Genesis block	Mainnet-specific	Testnet-specific
`IsStandard()` check to ensure a transaction is standard	Enabled	Disabled

Various faucet services are also available for the Bitcoin test network. These services are used to get some test Bitcoin for testnet accounts. A list of faucets is available here on the Bitcoin wiki: `https://en.bitcoin.it/wiki/Testnet#Faucets`. The availability of test coins is very useful for experimentation on the testnet.

The command line to start up the Bitcoin testnet is as follows. To run the Bitcoin daemon for the testnet:

```
$ bitcoind --testnet -daemon
```

To run the Bitcoin command-line interface:

```
$ bitcoin-cli --testnet <command>
```

To allow the Bitcoin GUI to run in the testnet:

```
$ bitcoin-qt –testnet
```

A sample run is shown here:

1. Start up the Bitcoin node in daemon (as a background process) mode on the testnet:

```
$ bitcoind --testnet -daemon
Bitcoin server starting
```

2. Check the number of blocks and the difficulty. Note that there is a long list of various commands that the Bitcoin client supports. This is just an example to show how the Bitcoin command-line interface works:

```
$ bitcoin-cli --testnet getmininginfo
{
  "blocks": 566251,
  "difficulty": 400.6820950060902,
  "networkhashps": 572058533067.9225,
  "pooledtx": 0,
  "chain": "test",
  "warnings": ""
}
```

3. A complete list of commands can be obtained by running the following command:

```
$ bitcoin-cli --testnet help
```

The preceding command output shows various command-line options available in `bitcoin-cli`, the Bitcoin command-line interface. These commands can be used to query the blockchain, send transactions, and control the local node.

4. We now can stop the Bitcoin daemon by using the command shown here:

```
$ bitcoin-cli --testnet stop
Bitcoin server stopping
```

With this, we have completed a basic introduction to the Bitcoin testnet. We will do some more experimentation with this shortly, but first, we will look at another mode in which the Bitcoin node can run and that is especially useful for testing purposes.

Starting up a node in regtest

Regtest mode (regression testing mode) can be used to create a local private blockchain for testing purposes. In this mode, the user can control block generation for experimentation and testing, and a number of blocks with no value can be generated. It effectively creates a locally isolated new bitcoin blockchain for testing purposes.

The following commands can be used to start up a node in regtest mode:

1. Start up the Bitcoin daemon in regtest mode:

```
$ bitcoind -regtest -daemon
Bitcoin server starting
```

2. Check the balance:

```
$ bitcoin-cli -regtest getbalance
0.00000000
```

3. Generate blocks and addresses:

```
$ bitcoin-cli -regtest generatetoaddress 200 $(bitcoin-cli -regtest
getnewaddress)
[
  "366fce3c35031eaa3b085ae7d2631cb5b212bac7e3447bd8ffddb17ef97569c4
.
.
.
"33361a74d2586259d69a724921ff7b931cc6c95bd52f09fc05a4b8905695384f"
]
```

The reason why we generated 200 blocks in the preceding command is that on a regtest, a block must have 100 confirmations before the associated reward can be utilized. Therefore, we must generate more than 100 blocks to get access to this reward. In this command, we have generated 200 blocks, which will generate 5,000 bitcoins due to the hardcoded miner reward of 50 bitcoins.

4. Now, we can get the balance by running the following command:

```
$ bitcoin-cli -regtest getbalance
5000.00000000
```

5. Run a command, for example, `getmininginfo`:

```
$ bitcoin-cli -regtest getmininginfo
{
  "blocks": 200,
  "currentblockweight": 4000,
  "currentblocktx": 0,
  "difficulty": 4.656542373906925e-10,
  "networkhashps": 12,
  "pooledtx": 0,
  "chain": "regtest",
  "warnings": ""
}
```

6. We can also get information about the blockchain by using the following command:

```
$ bitcoin-cli -regtest getblockchaininfo
{
  "chain": "regtest",
  "blocks": 200,
  "headers": 200,
  "bestblockhash":
"1cafd1e540b6772f4fe4ab561def0de69945f84e701e5fffa8426ea572d3769b",
  "difficulty": 4.656542373906925e-10,
  "mediantime": 1577225980,
  .
  .
  .
```

Note that the complete output is not shown here due to its long length, but it is enough to explain the concept.

7. Stop the Bitcoin daemon:

```
$ bitcoin-cli -regtest stop
Bitcoin server stopping
```

If you want to delete the previous regtest node and start a new one, simply delete the directory named `regtest` under your computer's `$HOME` directory. On a Mac (macOS), it is located at `/$HOME/Library/Application Support/Bitcoin`.

After deleting the `regtest` directory, run the command shown in the first step again in this section to create a new `regtest` environment.

In this section, we covered how to start up a Bitcoin node in test and development (regtest) modes and how to interact with the Bitcoin blockchain using `bitcoin-cli`, the command-line tool for the Bitcoin client. Next, we will experiment further with some Bitcoin commands and interfaces.

Experimenting further with bitcoin-cli

As we've seen so far, `bitcoin-cli` is a powerful and feature-rich command-line interface available with the Bitcoin Core client and can be used to perform various functions using the RPC interface provided by the Bitcoin Core client.

We will now see how to send bitcoins to an address using the command line. For this, we will use the Bitcoin command-line interface on the Bitcoin regtest:

1. Generate a new address using the following command:

```
$ bitcoin-cli -regtest getnewaddress
2NC31WFFRwRkwd3S4TpyjN5GGDY7E63GSVd
```

2. Send 20 BTC to the newly generated address:

   ```
   $ bitcoin-cli -regtest sendtoaddress \
   2NC31WFFRwRkwd3S4TpyjN5GGDY7E63GSVd 20.00
   ```

   The output of this command will show the transaction ID, which is:

   ```
   a83ff460a32f29387d531f19e7092a5dcf6ce52d20931227447c0b9b7a5f2980
   ```

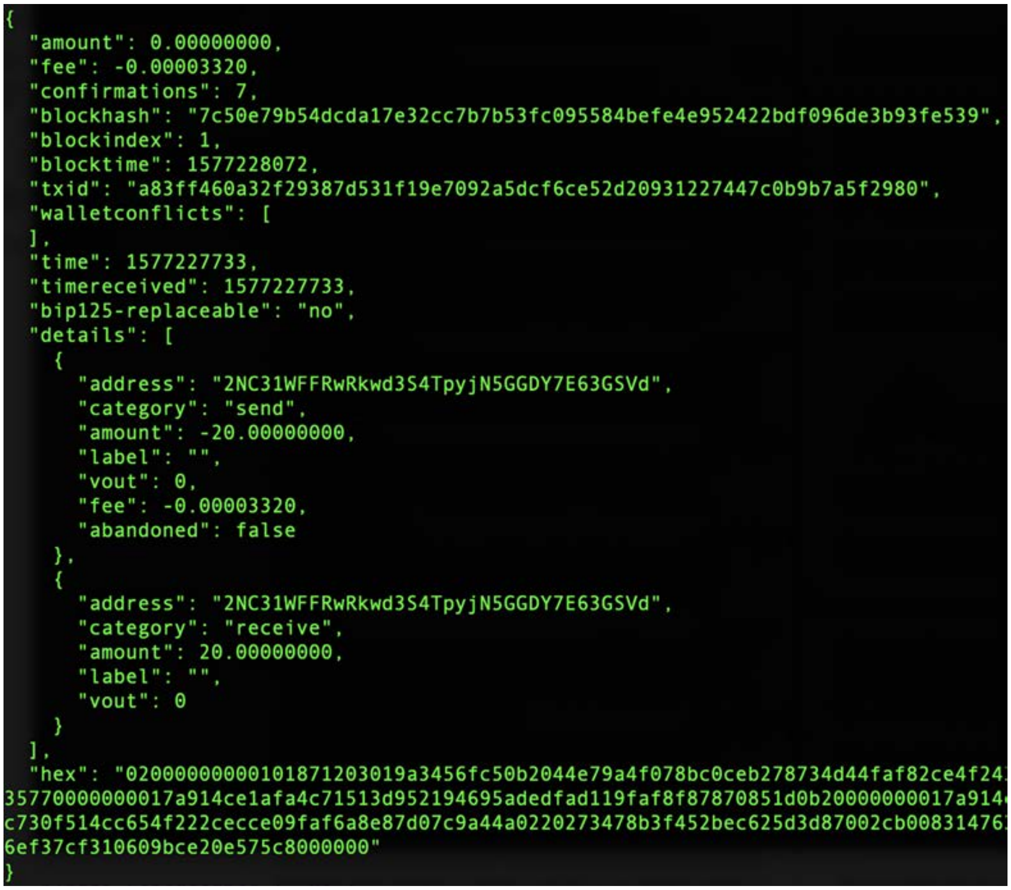
3. We can now generate a few more blocks to get some confirmation for this:

   ```
   $ bitcoin-cli -regtest generatetoaddress 7 $(bitcoin-cli -regtest
   getnewaddress)
   ```

4. We can also query the transaction information by using the following command:

   ```
   $ bitcoin-cli -regtest gettransaction \
   a83ff460a32f29387d531f19e7092a5dcf6ce52d20931227447c0b9b7a5f2980
   ```

   This will show an output similar to the one shown here. Note that we use the same transaction ID hash output that was generated in *step 2* previously:



```
{
  "amount": 0.00000000,
  "fee": -0.00003320,
  "confirmations": 7,
  "blockhash": "7c50e79b54dcda17e32cc7b7b53fc095584befe4e952422bdf096de3b93fe539",
  "blockindex": 1,
  "blocktime": 1577228072,
  "txid": "a83ff460a32f29387d531f19e7092a5dcf6ce52d20931227447c0b9b7a5f2980",
  "walletconflicts": [
  ],
  "time": 1577227733,
  "timereceived": 1577227733,
  "bip125-replaceable": "no",
  "details": [
    {
      "address": "2NC31WFFRwRkwd3S4TpyjN5GGDY7E63GSVd",
      "category": "send",
      "amount": -20.00000000,
      "label": "",
      "vout": 0,
      "fee": -0.00003320,
      "abandoned": false
    },
    {
      "address": "2NC31WFFRwRkwd3S4TpyjN5GGDY7E63GSVd",
      "category": "receive",
      "amount": 20.00000000,
      "label": "",
      "vout": 0
    }
  ],
  "hex": "020000000001018712030l9a3456fc50b2044e79a4f078bc0ceb278734d44faf82ce4f24
35770000000017a914ce1afa4c71513d952194695adedfad119faf8f87870851d0b20000000017a914
c730f514cc654f222cecce09faf6a8e87d07c9a44a0220273478b3f452bec625d3d87002cb00831476
6ef37cf310609bce20e575c8000000"
}
```

Figure 7.3: gettransaction output

So far, we've used `bitcoin-cli` on regtest. However, we can use `bitcoin-cli` on any Bitcoin network, for example, the testnet or the mainnet. We simply use the `bitcoin-cli` command without specifying any network to query the mainnet blockchain. Next, we will show a quick example of querying the Bitcoin mainnet blockchain. The Bitcoin client provides three methods for interacting with the blockchain, as listed here:

- Bitcoin **Command-Line Interface** (**CLI**)
- JSON-RPC interface
- HTTP REST interface

First, we'll see an example of `bitcoin-cli` querying the blockchain using the `getblock` method. We will then see how the same `getblock` method can be invoked using the JSON-RPC interface and the HTTP REST interface.

We will query the 100th block of the Bitcoin blockchain with hash `000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a`.

Using the Bitcoin command-line tool

We can use `bitcoin-cli` for the purpose of using the Bitcoin command-line tool, as shown here:

```
$ bitcoin-cli getblock \
"000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a"
```

The output of the preceding command, which shows the details of block hash `000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a`, is shown below:

```
{
  "hash": "000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a",
  "confirmations": 232839,
  "strippedsize": 215,
  "size": 215,
  "weight": 860,
  "height": 100,
  "version": 1,
  "versionHex": "00000001",
  "merkleroot":
"2d05f0c9c3e1c226e63b5fac240137687544cf631cd616fd34fd188fc9020866",
  "tx": [
    "2d05f0c9c3e1c226e63b5fac240137687544cf631cd616fd34fd188fc9020866"
  ],
  "time": 1231660825,
  "mediantime": 1231656204,
  "nonce": 1573057331,
  "bits": "1d00ffff",
  "difficulty": 1,
  "chainwork": "000000000000000000000000000000000000000000000000000000650065006
5",
  "nTx": 1,
  "previousblockhash":
"00000000cd9b12643e6854cb25939b39cd7a1ad0af31a9bd8b2efe67854b1995",
  "nextblockhash":
"00000000b69bd8e4dc60580117617a466d5c76ada85fb7b87e9baea01f9d9984"
}
```

The output shows several elements, including previous block hash, next block hash, transaction hashes included in the block, number of transactions, difficulty level, time, and some other data.

As an alternative to `bitcoin-cli`, we can query the blockchain using the JSON-RPC interface provided by the Bitcoin client as well. We'll show an example of that next.

Using the JSON-RPC interface

Now, we will run the same command but using the JSON-RPC. Note that we are using the Bitcoin mainnet for this example.

At a minimum, in order to use the JSON-RPC interface, we need to configure the RPC username and password in the `bitcoin.conf` file. We can easily do that. A sample configuration that will be used in this example is shown here:

```
$ cat bitcoin.conf
rpcuser=test1
rpcpassword=testpassword
```

We can use the `curl` command-line tool to interact with the JSON-RPC API, as shown here:

```
$ curl --user test1 --data-binary '{"jsonrpc": "1.0",
"id":"curltest", "method": "getblock", "params":
["000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a"] }' -H
'content-type: text/plain;' http://127.0.0.1:8332/
```

This will ask for the required password. Enter the password that has been set in the `bitcoin.conf` file:

```
Enter host password for user 'test1':
```

If the password is correct, after executing the command, the result shown here will be displayed in JSON format:

```
{"result":{"hash":"000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9
ebf9a715ed27fdb229a","confirmations":236138,"strippedsize":215,
"size":215,"weight":860,"height":100,"version":1,"versionHex":"00000001",
"merkleroot":"2d05f0c9c3e1c226e63b5fac240137687544cf631cd616fd34fd188fc9020866",
"tx":["2d05f0c9c3e1c226e63b5fac240137687544cf631cd616fd34fd188fc9020866"],
"time":1231660825,"mediantime":1231656204,"nonce":1573057331,"bits":
"1d00ffff","difficulty":1,"chainwork":"00000000000000000000000000000
0000000000000000000000006500650065","nTx":1,"previousblockhash":
"00000000cd9b12643e6854cb25939b39cd7a1ad0af31a9bd8b2efe67854b1995",
"nextblockhash":"00000000b69bd8e4dc60580117617a466d5c76ada85fb7b87e9bae
a01f9d9984"},"error":null,"id":"curltest"}
```

`curl` is an excellent command-line tool that is used to transfer data using URLs. It is commonly used to interact with REST APIs using HTTP. More information about `curl` is available at `https://curl.haxx.se`.

An example run will be shown next, which queries the same block that we queried in the previous command, but now using the HTTP REST interface.

Using the HTTP REST interface

Starting from Bitcoin Core client 0.10.0, the HTTP REST interface is also available. By default, this runs on the same TCP port (`8332`) as the JSON-RPC interface and requires no authentication. It is enabled either in the `bitcoin.conf` file by adding the `rest=1` option or on the `bitcoind` command line via the `-rest` flag.

We can use `curl` again for this purpose, as shown here:

```
$ curl http://localhost:8332/rest/
block/000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a.json
```

The output of the preceding command is shown here:

```
{"hash":"000000007bc154e0fa7ea32218a72fe2c1bb9f86cf8c9ebf9a715ed27fdb229a",
"confirmations":247383,"strippedsize":215,"size":215,"weight":860,"height":100,
"version":1,"versionHex":"00000001","merkleroot":"2d05f0c9c3e1c226e63b5fac240
137687544cf631cd616fd34fd188fc9020866","tx":[{"txid":"2d05f0c9c3e1c226e63b5
fac240137687544cf631cd616fd34fd188fc9020866","hash":"2d05f0c9c3e1c226e63b5
fac240137687544cf631cd616fd34fd188fc9020866","version":1,"size":134,"vsize":134,
"weight":536,"locktime":0,"vin":[{"coinbase":"04ffff001d014d",
"sequence":4294967295}],"vout":[{"value":50.00000000,"n":0,"scriptPubKey":
{"asm":"04e
70a02f5af48a1989bf630d92523c9d14c45c75f7d1b998e962bff6ff9995fc5bdb44f1793b37
495d80324acba7c8f537caaf8432b8d47987313060cc82d8a93 OP_CHECKSIG","hex":"4104e70
a02f5af48a1989bf630d92523c9d14c45c75f7d1b998e962bff6ff9995fc5bdb44f1793b37495d80
324acba7c8f537caaf8432b8d47987313060cc82d8a93ac","reqSigs":1,"type":"pubkey",
"addresses":["13A1W4jLPP75pzvn2qJ5KyyqG3qPSpb9jM"]}}],"hex":"01000000010000000
000000000000000000000000000000000000000000
000000000000ffffffff0704ffff001d014dffffffff
0100f2052a01000000434104e70a02f5af48a1989bf630d92523c9d14c45c75f7d1b998e962bff6ff
9995fc5bdb44f1793b37495d80324acba7c8f537caaf8432b8d47987313060cc82d8a93ac00000
000"}],"time":1231660825,"mediantime":1231656204,"nonce":1573057331,"bits":
"1d00ffff","difficulty":1,"chainwork":"000000000000000000000000000000000000
0000000000000006500650065","nTx":1,"previousblockhash":"00000000cd9b12643
e6854cb25939b39cd7a1ad0af31a9bd8b2efe67854b1995","nextblockhash":"000000
00b69bd8e4dc60580117617a466d5c76ada85fb7b87e9baea01f9d9984"}
```

With this, we have completed our basic introduction to the Bitcoin command-line tools and related interfaces. A complete introduction to all Bitcoin client RPC calls is not possible here. You are encouraged to check the comprehensive documentation of Bitcoin Core 0.21.0 RPC, which is available at `https://bitcoincore.org/en/doc/0.21.0/rpc/`.

Bitcoin programming

Bitcoin programming is a very rich field. The Bitcoin Core client exposes various JSON-RPC commands that can be used to construct raw transactions and perform other functions via custom scripts or programs. Also, the command-line tool `bitcoin-cli` is available, which makes use of the JSON-RPC interface and provides a rich toolset to work with Bitcoin.

These APIs are also available via many online service providers in the form of Bitcoin APIs, and they provide a simple HTTP REST interface. Bitcoin APIs, such as blockchain.info (`https://blockchain.info/api`), BitPay (`https://bitpay.com/api`), block.io (`https://www.block.io`), and many others, offer a myriad of options to develop Bitcoin-based solutions.

Various libraries are available for Bitcoin programming. A list is shown as follows. Those of you who are interested can explore the libraries further:

- **Libbitcoin**: Available at `https://libbitcoin.dyne.org/` and provides powerful command-line utilities and clients.
- **Pycoin**: Available at `https://github.com/richardkiss/pycoin`, this is a library for Python.
- **Bitcoinj**: This library is available at `https://bitcoinj.github.io/` and is implemented in Java.

There are many online Bitcoin APIs available; some commonly used APIs are listed as follows:

- `https://bitcore.io/`
- `https://bitcoinjs.org/`
- `https://blockchain.info/api`

As all APIs offer a similar type of functionality, it can get confusing to decide which one to use. It is also difficult to recommend which API is the best because all APIs are similarly feature-rich. One thing to keep in mind, however, is security. Therefore, whenever you evaluate an API for usage, in addition to assessing the offered features, also evaluate how secure the design of the API is.

Summary

In this chapter, we looked at Bitcoin payments and payment processors, along with some important Bitcoin innovations, which included topics such as BIPs and advanced Bitcoin protocols.

The chapter continued with an introduction to Bitcoin installation, followed by a discussion on source code setup and how to set up Bitcoin clients for various networks. After this, we examined various command-line options available in Bitcoin clients. Lastly, we saw which APIs are available for Bitcoin programming and the main points to keep in mind while evaluating APIs for usage.

In the next chapter, we'll introduce smart contracts, which are an integral element of programmable blockchains and allow programmers to write programs that are stored on blockchain and implement business logic on the chain.

Join us on Discord!